# TB2J

**Sep 06, 2021**

# Contents:

TB2J is a open source Python package for the automatic computation of magnetic interactions (including exchange and Dzyaloshinskii-Moriya) between atoms of magnetic crystals from density functional Hamiltonians based on Wannierfunctions or linear combination of atomic orbitals. The program is based on the Green's function method with thelocal rigid spin rotation treated as a perturbation. As input, the package uses the output of either Wannier90, whichis interfaced with many density functional theory packages, or of codes based on localised orbitals. A minimal userinput is needed, which allows for easy integration into high-throughput workflows. The source code can be found at https://github.com/mailhexu/TB2J. For questions please use the online forum at https://groups.google.com/g/tb2j, or send email to mailto:tb2j@googlegroup.com. More TB2J examples with full DFT/Wannier data can be found at https://github.com/mailhexu/TB2J_examples

CHAPTER 1

# Installation

## 1.1 Dependencies

- python (>=3.6)
- numpy
- scipy
- ASE (atomic simulation environment)
- matplotlib
- sisl (optional) for Siesta interface
- GPAW (optional) For gpaw interface

## 1.2 How to install

The most easy way to install TB2J is to use pip:

```
pip install TB2J
```

You can also download TB2J from the github page, and install with

```
python setup.py install
```

The –user option will help if there is permission problem.

It is suggested that it being installed within a virtual environment using e.g. pyenv or conda.

By default, TB2J only forces the non-optional dependencies to be installed automatically. The sisl package which is used to read the Hamiltonian from the Siesta or OpenMX output is needed, which can also be installed with pip. The GPAW-TB2J interface is through python directly, which of course requires the gpaw python package. The sisl and gpaw python package can be installed via pip, too.

# CHAPTER 2

## Conventions of Heisenberg Model

Before you use the TB2J output, please read very carefully this section. There are many conventions of the Heisenberg. We strongly suggest that you clearly specify the convention you use in any published work. Here we describe the convention used in TB2J.

The Heisenberg Hamiltonian contains four different parts and reads as

$$\begin{aligned} E = & -\sum_i K_i \vec{S}_i^2 \\ & -\sum_{i \neq j} \Big[ J_{ij}^{iso} \vec{S}_i \cdot \vec{S}_j \\ & + \vec{S}_i \mathbf{J}_{ij}^{ani} \vec{S}_j \\ & + \vec{D}_{ij} \cdot \left( \vec{S}_i \times \vec{S}_j \right) \Big], \end{aligned}$$

where the first term represents the single-ion anisotropy (SIA), the second the isotropic exchange, and the third term is the symmetric anisotropic exchange, where $\mathbf{J}^{ani}$ is a $3 \times 3$ tensor with $J^{ani} = J^{ani,T}$. The final term is the DMI, which is antisymmetric. Importantly, the SIA is not accessible from Wannier 90 as it requires separately the spin-orbit coupling part of the Hamiltonian. However, it is readily accessible from constrained DFT calculations.

We note that there are several conventions for the Heisenberg Hamiltonian, here we take a commonly used one in atomic spin dynamics: we use a minus sign in the exchange terms, i.e. positive exchange $J$ values favor ferromagnetic alignment. Every pair $ij$ is taken into account twice, $J_{ij}$ and $J_{ji}$ are both in the Hamiltonian. Similarly, both $uv$ and $vu$ are in the symmetric anisotropic term. The spin vectors $\vec{S}_i$ are normalized to 1, so that the parameters are in units of energy. The other commonly used conventions differ in a prefactor 1/2 or a summation over different $ij$ pairs only. The conversion factors to other conventions are given in the following table. For other conventions in which the spins are not normalized, the parameters need to be divided by $|\vec{S}_i \cdot \vec{S}_j|$ in addition.

CHAPTER 3

# Tutorial

## 3.1 Use TB2J with Wannier90

This tutorial uses cubic $SrMnO_3$ as an example to show how to calculate the exchange parameters for the Heisenberg model starting from density functional theory. First, the Hamiltonian in the basis of Wannier functions (WF) is constructed using Wannier90. Then, TB2J is used to calculate the exchange parameters. We assume that the reader has a basic knowledge of maximally localized WFs and the Wannier90 package (see Maximally localized Wannier Functions, Wannier90).

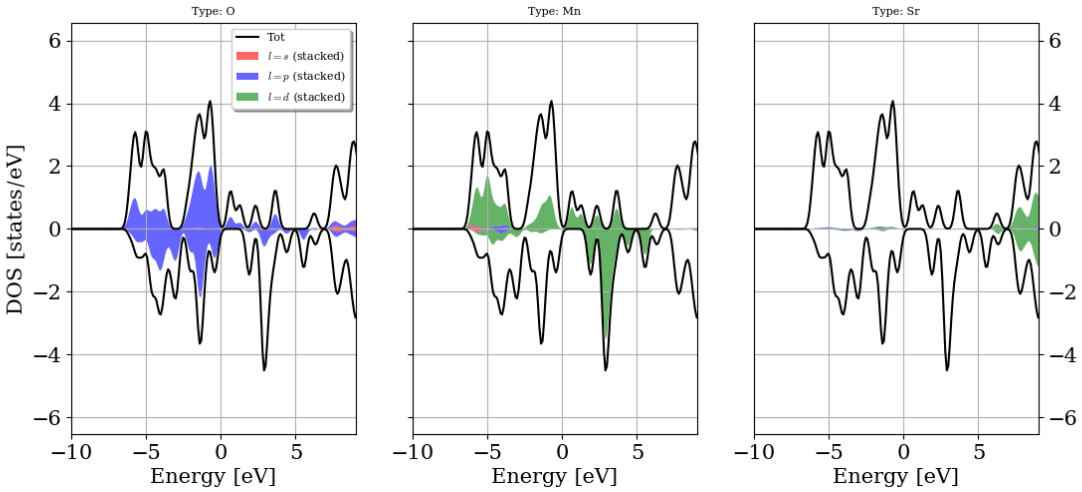
*Before beginning, you might consider to work in a subdirectory for this tutorial. Why not Work_tb2j?*

The input files for the tutorial can be found inside examples/abinit-w90/SrMnO3 in your TB2J directory. Please copy abinit.in, abinit.files and the three pseudopotential files (inside the psp directory) to *Work_tb2j*. You also need the two files abinito_w90_down.win and abinito_w90_up.win which provide additional input for Wannier90. The names of these two files are _w90_.win with the prefix being given in the forth line of the .files file. Modify the .files file such that the entries match the location of your files.

### 3.1.1 Step 0: Find the orbitals and energy range to be used in the Wannier Function Hamiltonian.

Before we can construct the Hamiltonian in the basis of the Wannier functions, we need to determine which orbitals to include in the construction. We need to include the orbitals with energies around the Fermi energy ($E_F$). Since we are interested in calculating exchange parameters we need to include spin as a degree of freedom in the calculation and select the magnetic orbitals and all orbitals that overlap with them. To determine the orbitals and energy range, we calculate either the density of states or the band structure of the system. For $SrMnO_3$ the density of states is given in the figure below.

As we can see, the Mn 3d and O 2p orbitals should be included into the WF Hamiltonian. The Sr 4d orbitals are too high in energy, so we exclude them from the WF Hamiltonian.




### 3.1.2 Step 1: Construct WF Hamiltonian from DFT.

The Wannier90 code makes use of two energy windows to disentangle the bands. An outer window (the disentangle window), which contains all the required orbitals, and an inner window (the frozen window), which only contain the required orbitals, should be provided. From the DOS we find that all the Mn 3d and O 2p bands are between -10 and 10 eV, the Sr 4d bands above 6 eV, which should be excluded from the frozen window. Thus we can select the energy window (-10, 10) eV and the frozen window of (-9, 5) eV. Note that the energy defined in Wannier90 is not relative to $E_F$, so we need to add the Fermi energy (here: 6.15 eV) to the energies. We use Mn d and O p orbitals as an initial guess for the WFs. This information can be found in the .win files

```
# Energy windows (Fermi energy is 6.15 eV)
dis_win_min = -3.85
dis_win_max = 16.15
dis_froz_min = 1.15
dis_froz_max = 11.15

begin projections
Mn: d
O : p
end projections
```

For a detailed explanation of the input variables for Wannier90 please see Wannier90. For our purpose, it is important to write out the Hamiltonian and the centers of the Wannier functions.

```
# write the postitions of WF
write_xyz = true

# write the WF Hamiltonian (Note for W90 version<2.1, it is hr_plot)
write_hr = true
```

The following lines need to be added to the abinit input file to generate WFs.

```
prtwant 2     # enable wannier90
w90iniprj 2   # use projection to orbitals instead of random.
w90prtunk 0   # use 1 if you want to visualize the WF's later.
```

Now you can run

```
abinit < abinit.files > log 2> err
```

which generates the files below for spin up, and the same set for spin down

```
abinito_w90_up_hr.dat abinito_w90_up_centres.xyz   abinito_w90_up.wout
```

The .dat file contains the Hamiltonian, the .xyz file contains the Wannier centers. The .wout file has a summary of the process of running Wannier90 and will be used to calculate the exchange parameters.

If you're using Wannier90 version < 3.0, the spin down files are not automatically generated due to a bug. To get the files, the following command is needed:

```
wannier90.x abinito_w90_down
```

To get localized WFs can be tricky sometimes. It is necessary to check if the WFs are localized by looking at the .wout file. For example, we have

```
Final State
 WF centre and spread    1  (  1.904992,  1.904992,  1.904992 )     0.50185811
 WF centre and spread    2  (  1.904992,  1.904992,  1.904992 )     0.48650086
 WF centre and spread    3  (  1.904992,  1.904992,  1.904992 )     0.48650086
 WF centre and spread    4  (  1.904992,  1.904992,  1.904992 )     0.50185997
 WF centre and spread    5  (  1.904992,  1.904992,  1.904992 )     0.48650084
 WF centre and spread    6  (  1.904992,  1.904992, -0.000000 )     0.74591265
 WF centre and spread    7  (  1.904992,  1.904992,  0.000000 )     0.96557405
 WF centre and spread    8  (  1.904992,  1.904992, -0.000000 )     0.96557405
 WF centre and spread    9  ( -0.000000,  1.904992,  1.904992 )     0.96557489
 WF centre and spread   10  ( -0.000000,  1.904992,  1.904992 )     0.74589254
 WF centre and spread   11  (  0.000000,  1.904992,  1.904992 )     0.96557379
 WF centre and spread   12  (  1.904992,  0.000000,  1.904992 )     0.96557489
 WF centre and spread   13  (  1.904992, -0.000000,  1.904992 )     0.96557379
 WF centre and spread   14  (  1.904992, -0.000000,  1.904992 )     0.74589254
 Sum of centres and spreads ( 20.954915, 20.954915, 20.954915 )    10.49436382
```

Usually, 3d orbitals have a spread of less than 1 , and the O 2p orbitals have a spread of less than 2 .

### 3.1.3 Step 2: Run TB2J

Before running TB2J, an extra file, which contains the atomic structure, needs to be prepared. It can be either a VASP POSCAR file. (For abinit, the abinit.in file is also fine if no fancy feature is used, like use of *, or units. POSCAR files are recommended because they are simple. Note that the file extension are used to identify the format, for example, Quantum ESPRESSO input should be name with *.pwi) The supported file format are can be found on the list in: https://wiki.fysik.dtu.dk/ase/ase/io/io.html

(From version 0.6.2 this file is no more necessary as TB2J can read the atomic structures from the Wannier90 .win file). The –posfile option will still be used by default if it is specified.)

With the WF Hamiltonian generated, we can calculate the exchange parameters now. In the scripts directory inside your TB2J directory you find the wann2J.py script. Please make sure that it is executable and issue the command

```
wann2J.py --posfile abinit.in --efermi 6.15 --kmesh 4 4 4 --elements Mn --prefix_up␣
↪abinito_w90_up --prefix_down abinito_w90_down  --emin -10.0 --emax 0.0
```

The parameters are:

- efermi: Fermi energy in eV
- kmesh: k-point mesh. Default is 5 5 5

- elements: the magnetic elements
- prefix_up: prefix for spin up channel of the Wannier90 output
- prefix_down: prefix for spin down channel of Wannier90 output.
- emin: the lower limit of the electron energy. (in eV, relative to Fermi energy.)
- emax: the upper limit of the electron energy. Should be close to zero.

Now we should have the files containing the J parameters in the TB2J_results directory.

```
TB2J_results/
├── exchange.txt
├── Multibinit
│   ├── exchange.xml
│   ├── mb.files
│   └── mb.in
├── TomASD
│   ├── exchange.exch
│   └── exchange.ucf
└── Vampire
    ├── input
    ├── vampire.mat
    └── vampire.UCF
```

- exchange.txt: A human readable file.
- Multibinit directory: the files file, input file and xml file, which can be used as templates to run spin dynamics in Multibinit.
- The input for a few spin dynamics codes (Tom's ASD, and Vampire) are also included.

### 3.1.4 Noncollinear calculation

For calculations with non-collinear spin, the –spinor option should be used. It is also necessary to specify whether in the Hamiltonian the order of the basis, either group by spin (orb1_up, orb2_up, ... orb1_down, orb2_down, ...) or by orbital (orb1_up, orb1_down, orb2_up, orb2_down,...), with the –groupby option (either spin or orbital). The –prefix_spinor option is used to specify the prefix of the Wannier90 outputs. Here is an example of the command:

```
wann2J.py --spinor --groupby spin --posfile abinit.in --efermi 6.15 --kmesh 4 4 4 --
↪elements Mn --prefix_spinor abinito
```

## 3.2 Use TB2J with Siesta

In this tutorial we will learn how to use TB2J with Siesta. First we calculate the isotropic exchange in bcc Fe. Then we calculate the isotropic exhange, anisotropic exchange and Dzyanoshinskii-Moriya interaction (DMI) parameters from a calculation with spin-orbit coupling enabled. Before running this tutorial, please make sure that the sisl package is installed.

### 3.2.1 Collinear calculation without SOC

Let's start from the example of BCC Fe. The input files used can be found in the examples/Siesta/bccFe directory.

First, we do a siesta self consistent calculation with the BCC Fe primitive cell of bcc Fe has only one Fe atom. In the example, we use the pseudopotential from the PseudoDojo dataset in the psml format. Note that at the moment

(December 2020) the psml support is not yet in the main branch of Siesta. We provide the .psf pseudopotential converted from the .psml file. To use the psml format, one could find the Siesta psml branch on gitlab (https://gitlab.com/siesta-project/siesta/-/tree/psml-support). A default double zeta polarized (DZP) basis set. We need to save the electronic Kohn-Sham Hamiltonian in the atomic orbital basis set with the options:

```
CDF.Save True
SaveHS True
Write.DMHS.Netcdf   True
```

After that, we will have the files siesta.nc and DMHS.nc file, which contains the Hamiltonian and overlap matrix information.

Now we can run the siesta2J.py command to calculate the exchange parameters:

```
siesta2J.py --fdf_fname siesta.fdf --elements Fe --kmesh 7 7 7
```

This first read the siesta.fdf, the input file for Siesta. It then read the Hamiltonian and the overlap matrices, calculate the J with a $7 \times 7 \times 7$ k-point grid. This allows for the calculation of exchange between spin pairs between $i$ and $j$ in a $7 \times 7 \times 7$ supercell, where $i$ is fixed in the center cell.

### 3.2.2 Non-collinear calculation

The anisotropic exchange and the DMI parameters can be calculated with non-collinear DFT calculation. The procedure is almost the same as in the collinear calculation except that the parameters for non-collinear calculation must be set in the Siesta input (Spin should be set to non-colinear or spin-orbit).

## 3.3 Use TB2J with OpenMX

In this tutorial we will learn how to use TB2J with OpenMX with the example of cubic SrMnO3. The example input files can be found in the examples directory of

> The interface to OpenMX is distributed as a plugin to TB2J called TB2J OpenMX under the GPL license, which need to be installed separately, because code from OpenMX which is under the GPL license is used in the parser of OpenMX files.

### 3.3.1 Install TB2J-OpenMX

```
pip install TB2J-OpenMX
```

### 3.3.2 running TB2J

In the DFT calculation, the "HS.fileout on" options should be enabled, so that the Hamiltonian and the overlap matrices are written to a ".scfout" file. Then we can run the command openmx2J.py. The necessary input are the path of the calculation, the prefix of the OpenMX files, and the magnetic elements:

```
openmx2J.py -- prefix openmx --elements Fe
```

openmx2J.py then read the openmx.xyz and the openmx.scfout files from the OpenMX output, and output the results to TB2J_results.

## 3.4 Parameters in calculation of magnetic interaction parameters

### 3.4.1 List of parameters:

The list of parameter can be found using:

```
wann2J.py --help
```

or

```
siesta2J.py --help
```

The parameter will be explained in the following text.

- kmesh: Three integers to specify the size of a Monkhorst-Pack mesh. This is the mesh of k-points used to calculate the Green's functions. The real space supercell in which the magnetic interactions are calculated, has the same size as the k-mesh. For example, a $7 \times 7 \times$ k-mesh is linked with a $7 \times 7 \times$ supercell, the atom $i$ resides in the center cell, whereas the $j$ atom can be in all the cells in the supercell.
- efermi: The Fermi energy in eV. For insulators, it can be inside the gap. For metals, it should be the same as in the DFT calculation. Due to the different algorithms in the integration of the density, the Fermi energy could be slightly shifted from the DFT value.
- emin, emax, and nz: During the calculation, there is a integration $\int_{emin}^{emax} d\epsilon$ calculation. The emin and emax are relative values to the Fermi energy. The emax should be close to 0. It can be used to adjust the integration if the charge . The emin should be low enough so that all the electronic states that affect the magnetic interactions are integrated. The nz is the number of steps in this integration.
- rcut: rcut is the cutoff distance between two ion pairs between which the magnetic interaction parameters are calculated. By default, all the pairs inside the supercell defined by the kmesh
- exclude_orbs: the indeces of orbitals, whose contribution will not be counted in the magnetic interaction. It is a list of integers. The indices are zero based.

## 3.5 Averaging multiple parameters

As discussed in the previous section, the $z$ component of the DMI, and the $xz$, $yz$, $zz$, $zx$, $zy$, components of the anisotropic exchanges are non-physical, and an $xyz$ average is needed to get the full set of magnetic interaction parameters if the spins are all along $z$. In this case, scripts to rotate the structure and merge the results are provided, they are named TB2J_rotate.py and TB2J_merge.py. The TB2J_rotate.py reads the structure file and generates three files containing the $z \rightarrow x$, $z \rightarrow y$,and the non-rotated structures. The output files are named atoms_x, atoms_y, atoms_z. A large number of output file formats is supported thanks to the ASE library and the format of the output structure files is provided using the –format parameter. An example for using the rotate file is:

```
TB2J_rotate.py BiFeO3.vasp --format vasp
```

---

**Note:** Some file format does not include the cell orientation, e.g. the cif file. They should not be used as the output format.

---

The user has to perform DFT single point energy calculations for these three structures in different directories, keeping the spins along the $z$ direction, and run TB2J on each of them. After producing the TB2J results for the three rotated structures, we can merge the DMI results with the following command by providing the paths to the TB2J results of the three cases:

```
TB2J_merge.py BiFeO3_x BiFeO3_y BiFeO3_z --type structure
```

Note that the whole structure are rotated w.r.t. the laboratory axis but not to the cell axis. Therefore, the k-points should not be changed in both the DFT calculation and the TB2J calculation.

A new TB2J_results directory is then made which contains the merged final results.

Another method is to do the DFT calculation with spins along the $x$, $y$ and $z$ axis, respectively, and then merge the result with:

```
TB2J_merge.py BiFeO3_x BiFeO3_y BiFeO3_z --type spin
```

## 3.6 The ligand spin problem: downfolding the Heisenberg Hamiltonian

In some structures, the spin magnetic momentum on the ligand are . They are however, not due to the spin splitting of the ligand atom, but the hybridization to the orbitals of surrounding atoms (often transitional metals). The refore, the spin will move with the

In these cases, a "downfolding" method could be used to used to get an effective Heisenberg model with only the transitional metal spins as independent variables from the exchange parameters with both transitional metal spins and ligand spins.

### 3.6.1 Usage:

To do the downfolding, one has to first generate the Heisenberg Hamiltonian with both the transitional metal and the ligands as magnetic elements, e.g. in CrI3,

```
wann2J.py --elements Cr I ....
```

The result is saved to TB2J_results directory. Then run the downfolding to get the Cr only effective exchange parameters, e.g.

```
TB2J_downfold.py --inpath TB2J_results --outpath TB2J_results_downfold --metals Cr --
↪ligands I
```

will generate the downfolded result to TB2J_results_downfold.

## 3.7 Decompose the exchange into orbital contributions.

The exchange J between two atoms can be decomposed into the sum of all the pairs of orbitals. For two atoms with m and n orbitals respectively, the decomposition can be written as a m b n matrix.

Here is an example:

Since version 0.6.4, the name of the orbitals are written in the Orbital contribution section. With the Wannier90 input, the names of these orbitals are not known, thus are named as orb_1, orb_2, etc. One can search the Wannier initial projectors to see what are these orbitals.

Here is an example of cubic SrMnO3,

```
================================================================================================
Orbitals used in decomposition:
The name of the orbitals for the decomposition:
Mn1 : ['orb_1', 'orb_2', 'orb_3', 'orb_4', 'orb_5']

================================================================================================
Exchange:
    i      j          R        J_iso(meV)          vector          distance(A)
------------------------------------------------------------------------------------------
↪--
   Mn1    Mn1    (  0,    0,    1) -7.1027    ( 0.000,  0.000,  3.810)  3.810
J_iso: -7.1027
Orbital contributions:
 [[ 3.184 -0.    -0.     0.    -0.   ]
 [-0.    -5.06   0.    -0.     0.   ]
 [-0.     0.    -5.06  -0.     0.   ]
 [ 0.    -0.    -0.    -0.121 -0.   ]
 [-0.     0.     0.     0.    -0.047]]
```

One can see that the contribution from the (orb_1, orb_1) pair is 3.184, and that from (orb_2, orb_2) is -5.06. Searching the wannier90 input, we can find that the orb1 and orb_2 are the dz2 and dxz orbitals, respectively.

For Siesta input, the names are known, and printed in the "orbitals use in decomposition" section. They are not the exactly the name of the siesta basis (like 3dxyZ1). For example, with a double-zeta basis set, the a 3dxy orbital might be splitted into 3dxyZ1 and 3dxyZ2. The contribution of them are summed up to make it more concise. Often, the contribution from some orbitals are negligible. In the siesta2J.py command, it's possible to specify the orbitals to be considered in the decompostion. For example, if only the 3dxy contribution of Cr is needed, one can write

```
siesta2J.py --elements Fe_3d  ....
```

If both the 3d and 4s are to be considered, one can write:

```
siesta2J.py --elements Fe_3d_4s  ....
```

For example, the bcc Fe, when only the 3d orbitals are turned on, we get:

```
================================================================================================
Orbitals used in decomposition:
The name of the orbitals for the decomposition:
Fe1 : ('3dxy', '3dyz', '3dz2', '3dxz', '3dx2-y2')

================================================================================================
Exchange:
    i      j          R        J_iso(meV)          vector          distance(A)
------------------------------------------------------------------------------------------
↪--
   Fe1    Fe1    ( -1,    0,    0) 17.6873    (-2.467,  0.000,  0.000)  2.467
J_iso: 17.6873
Orbital contributions:
 [[11.462 -0.     0.297 -0.     0.096]
 [-0.     3.69  -0.    -0.214 -0.   ]
 [-0.163 -0.     3.215 -0.    -3.262]
 [-0.     0.396 -0.    11.451 -0.   ]
 [-0.055  0.    -3.263  0.    -4.248]]
```

where we can find in the $x$ direction, the dxy and dxz orbitals contribute mostly to the ferromagnetic interaction, whereas the 3dx2-y2 contribution is antiferromagnetic.

Note that the option for selection of orbitals is not available in the Wannier90 interface, as the informations for labelling the orbitals are not included in the Wannier90 output (sometimes it is even not possible to do so as the Wannier functions are not necessarily atomic-orbital like).

## 3.8 The output of TB2J

In the following we describe the output files which TB2J produces. By running wann2J.py or siesta2J.py, a directory with the name TB2J_results (or the directory specified by the –output_path) will be generated, which contains the following output files:

- exchange.out: A human readable output file, which summarizes the results.
- Multibinit: A directory containing output which can be read directly by the Multibinit code.
- The exchange.out file contains three sections: cell, atoms and exchange. The cell section contains the lattice parameter matrix. The atoms section contains the positions, charges (for verification) and magnetic moments of the atoms: see example below

```
================================================================================================
Information:
Exchange parameters generated by TB2J 0.2.5.
================================================================================================
Cell (Angstrom):
0.030   3.950   3.950
3.950   0.030   3.950
3.950   3.950   0.030

================================================================================================
Atoms:
(Note: charge and magmoms only count the wannier functions.)
Atom_number       x          y          z      w_charge    M(x)       M(y)       M(z)
Bi1             0.2413     0.2413     0.2413     2.1538   -0.0015     0.0000    -0.0044
Bi2             4.2060     4.2060     4.2060     2.1538    0.0000     0.0000     0.0044
Fe1             2.0165     2.0165     2.0165     6.3564   -0.0260     0.0000     3.7927
Fe2             5.9812     5.9812     5.9812     6.3564   -0.0176     0.0000    -3.7927
O1              5.5238     2.1558     3.9388     4.8306   -0.0013     0.0000    -0.0705
.....
Total                                            46.0038   -0.0493     0.0000    -0.0000

================================================================================================
Exchange:
i           j           R       J_iso(meV)       vector         distance(A)
------------------------------------------------------------------------------
↪------
Fe2         Fe1    (0,   1,   1)  -28.9371 ( 3.934,  0.015,  0.015)  3.934
J_iso: -28.9371
[Testing!] Jprime: -59.620,  B: -15.342
[Testing!] DMI: (-0.5191  1.3581  0.1090)
[Testing!] J_ani:
[[ 0.    -0.002  0.047]
[-0.002  0.    -0.154]
[ 0.047 -0.154  0.   ]]
```

Here, the charge and magnetic moment of each atom are only integrated with the WFs attached to this atom. As such they can differ from the quantities coming from the direct DFT output, as not all bands are used in the construction of WFs. The WF charges should be integers, for LCAO the values depend on the band energy cutoffs. In addition, the exclusion of very deep lying levels from the calculation of $J$ can also lead to deviations in the charges which

might appear both for WFs and for LCAO. Another source of difference between the TB2J charges and magnetic moments and the DFT ones is the integration volume around the atoms, which is not necessarily the same. However, for localized $d$ and $f$ orbitals the magnetic moments should be close to their DFT counterparts, for TB2J to yield correct results for the parameters. Large differences between the TB2J and DFT values indicate that something may have gone wrong: either in in the contour integration $\int^{E_F} d\epsilon$ used in TB2J, or in the construction of the Wannier functions (incorrect wannierization process or too small WF basis set). Often, it comes from excluding an orbital that is important for the magnetic interaction in the studied system. In the case of metallic system, the Fermi energy might have to be slightly shifted with respect to the DFT reference due to different numerical method used in the integration of charge density.

Each pair of atoms is labeled by three parameters, the index $i$, $j$ and $R$, where $i$ and $j$ are the indices in the unit cell. The vector $\vec{R}$ specifies the cell the atom $j$ is translated to, i.e. the reduced positions of the two atoms are $\vec{r}_i$ and $\vec{r}_j + \vec{R}$, respectively. By default, the interaction is calculated within a supercell corresponding to the k-mesh. For example, with a $7 \times 7 \times 7$ k-mesh, all $ij$ pairs will be produced for the spin labeled $i$ in the center cell of a $7 \times 7 \times 7$ supercell. With the rcut flag, only the parameters for $ij$ pairs within a distance of rcut are calculated. The exchange parameters are reported as follows: magnetic atom $i$ connected with magnetic atom $j$, $R$ is the lattice vector between the unit cells containing $i$ and $j$, the value of $J$ for this pair of magnetic atoms in meV, the vector connecting them, and the distance between the pair of atoms. If SOC is enabled, the DMI and anisotropic $J^{ani}$ parameters are given in addition. The DMI vectors $\vec{D}$ and the anisotropic $J^{ani}$ are printed as vectors and matrices, respectively.

Apart from the main exchange.out file, TB2J delivers several other outputs, which provide the input for spin dynamics (SD) and Monte Carlo (MC) simulations. TB2J is interfaced with several SD and MC codes. It has native support to the Multibinit code delivered as part of the Abinit code since version 9.0 . The TB2J_results/Multibinit directory contains the templates of input files for this code. One can usually run spin-dynamics with slight or no modification of these files. “Testing” inputs are also generated for Vampire and Thomas Ostler’s GPU-ASD code.

CHAPTER 4

# Applications

## 4.1 Magnon band structure from TB2J output

In this section we show an application of calculating the magnon band structure using the TB2J results.

There is a script within the TB2J package: TB2J_magnon.py, which can be used to plot the magnon band structure. We can show its usage by:

```
TB2J_magnon.py --help

usage: TB2J_magnon.py [-h] [--fname FNAME] [--qpath QPATH]  [--figfname FIGFNAME] [--
↪show]

TB2J_magnon: Plot magnon band structure from the TB2J magnetic interaction parameters

optional arguments:
  -h, --help     show this help message and exit
  --fname FNAME  exchange xml file name. default: exchange.xml
  --qpath QPATH  The names of special q-points. If not given, the path will be␣
↪automatically choosen. See https://wiki.fysik.dtu.dk/ase/ase/dft/kpoints.html for␣
↪the table of special kpoints and the default path.
  --figfname FIGFNAME  The file name of the figure. It should be e.g. png, pdf or␣
↪other types of files which could be generated by matplotlib.
  --show         whether to show magnon band structure.
```

The input file specified to the –fname parameter is by default exchange.xml file, which is the output in the Multibinit xml format, as can be found in the TB2J_results/Multibinit directory.

Here we take the BCC Fe as an example. After we have the TB2J_results directory, we can go to the Multibinit directory and find the exchange.xml file. The q-path can be generated automatically from the atomic structure information, using the [ASE][https://wiki.fysik.dtu.dk/ase/ase] library. We need to specify a path of q-points if the default is not what we want. The default k-path and the labels of the special kpoints can be found at this page. For example, we want a path of "Gamma-N-P-Gamma-H-N" instead of the default, it can be given by:

```
TB2J_magnon.py --qpath GNPGHN --figfname magnon.png --show
```

The magnon band structure is then written to a By using the –show parameter, the band structure is shown on screen.

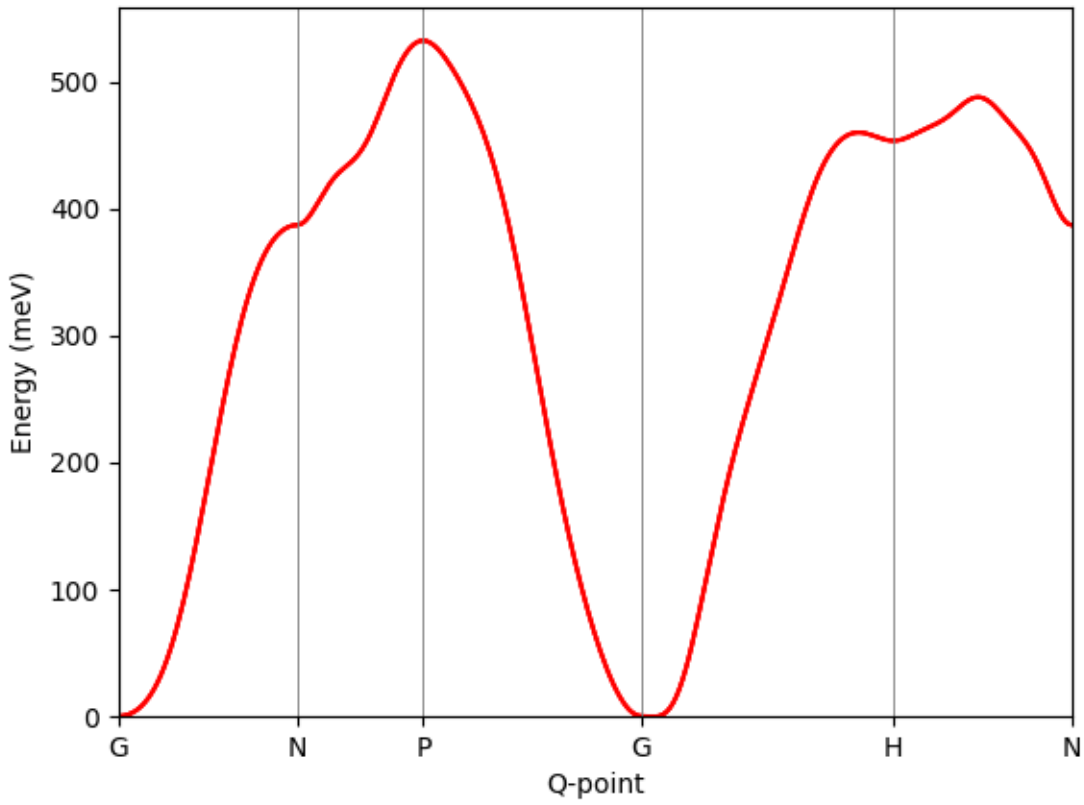



Fig. 1: exchange_magnon

## 4.2 Application: Spin model in MULTIBINIT

The output of TB2J can be directly readed by several atomistic spin dynamics and Monte Carlo code. Here we show how to use MULTIBINIT, a second principles code which can do spin dynamics. It is a part of the ABINIT package since version 9.0. The tutorial of the spin dynamics can be found on MULTIBINIT tutorial page.

Here we briefly describe how to run spin dynamics with MULTIBINIT from the TB2J outputs to with an example of BiFeO3. The example files can be found from the examples/Siesta/BiFeO3 directory.

In the TB2J_results/Mutlbinit directory, there are three files: exchange.xml, mb.in, and mb.files file. The exchange.xml file contains the Heisenberg parameters, the mb.in file is an input to the MULTIBINIT code, in which the parameters for the spin dynamics are given:

```
prt_model = 0

#-------------------------------------------------------------
```

(continues on next page)

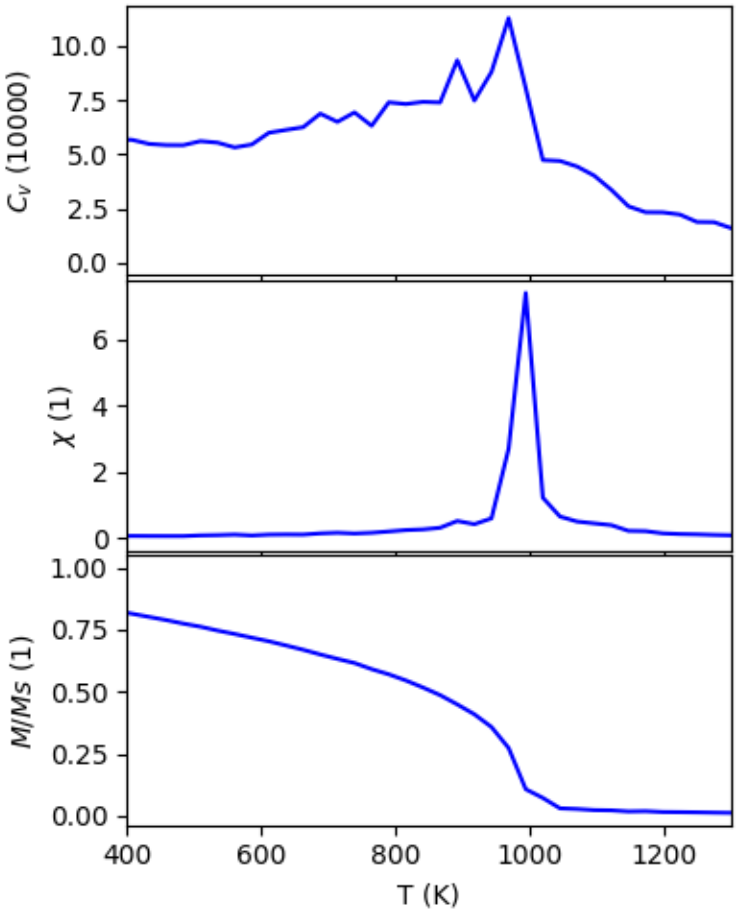
(continued from previous page)

```
#Monte carlo / molecular dynamics
#-------------------------------------------------------------
dynamics =  0    ! disable molecular dynamics

ncell =   28 28 28  ! size of supercell.
#-------------------------------------------------------------
#Spin dynamics
#-------------------------------------------------------------
spin_dynamics=1  ! enable spin dynamics
spin_mag_field= 0.0 0.0 0.0   ! external magnetic field
spin_ntime_pre = 10000          ! warming up steps.
spin_ntime =10000             ! number of steps.
spin_nctime=100               ! number of time steps between two nc file write
spin_dt=1e-15 s               ! time step.
spin_init_state = 1           ! FM initial state. May cause some trouble

spin_temperature=0.0

spin_var_temperature=1        ! switch on variable temperature calculation
spin_temperature_start=0      ! starting point of temperature
spin_temperature_end=1300      ! ending point of temperature.
spin_temperature_nstep= 52     ! number of temperature steps.

spin_sia_add = 1              ! add a single ion anistropy (SIA) term?
spin_sia_k1amp = 1e-6         ! amplitude of SIA (in Ha), how large should be used?
spin_sia_k1dir = 0.0 0.0 1.0  ! direction of SIA

spin_calc_thermo_obs = 1      ! calculate thermodynamics related observables
```

We can modify the default supercell size (ncell) and the temperature range to do spin dynamics in a temperature from 0K to 1300K in order to calculate the Neel temperature. We can run

```
mpirun -np 4 multibinit < mb.files
```

to run the spin dynamics. A mb.out.varT file is then generated, which has the volume heat capacit $C_v$, magnetic susceptibility
$chi$, and normalized total magnetic moment. They can be plotted as function of temperature as below, from which we can find the Neel temperature.


10.0
7.5
5.0
2.5
0.0
$C_v$ (10000)
6
4
2
0
$\chi$ (1)
1.00
0.75
0.50
0.25
0.00
$M/M_S$ (1)
400
600
800
1000
1200
T (K)

# CHAPTER 5

# Extending TB2J

In this section we show how to extend TB2J to interface with other first principles or similar codes and to write the output formats useful for codes such as spin dynamics.

## 5.1 Interface TB2J with other first principles or similar codes.

To interface a DFT code with TB2J, one has only to implement a tight-binding-like model which has certain methods and properties implemented. TB2J make use of the duck type feature of python, thus any class which has these things can be plugged in. Then the object can be inputted to the TB2J.Exchange class.

The methods and properties of AbstractTB class is listed as below.

**class** TB2J.myTB.**AbstractTB**(*R2kfactor*, *nspin*, *norb*)

**HS_and_eigen**(*kpts*)

get Hamiltonian, overlap matrices, eigenvalues, eigen vectors for all kpoints.

**Param**

- kpts: list of k points.

**Returns**

- H, S, eigenvalues, eigenvectors for all kpoints
- H: complex array of shape (nkpts, nbasis, nbasis)
- S: complex array of shape (nkpts, nbasis, nbasis). S=None if the basis set is orthonormal.
- evals: complex array of shape (nkpts, nbands)
- evecs: complex array of shape (nkpts, nbasis, nbands)

**get_hamR**(*R*)
get the Hamiltonian H(R), array of shape (nbasis, nbasis)

**get_orbs**()
returns the orbitals.

**is_siesta = None**
$\alpha$ used in $H(k) = \sum_R H(R)\exp(\alpha k \cdot R)$, Should be $2\pi i$ or $-2\pi i$

**nbasis = None**
nbasis=nspin*norb

**norb = None**
number of orbitals. Each orbital can have two spins.

**nspin = None**
number of spin. 1 for collinear, 2 for spinor.

**xcart = None**
The array of cartesian coordinate of all basis. shape:nbasis,3

**xred = None**
The array of cartesian coordinate of all basis. shape:nbasis,3

To pass the tight-binding-like model to the Exchange class is quite simple, here I take the sisl interface as an example

```
# read hamiltonian using sisl
fdf = sisl.get_sile(fdf_fname)
H = fdf.read_hamiltonian()
# wrap the hamiltonian to SislWrapper
tbmodel = SislWrapper(H, spin=None)
# pass to ExchangeNCL
exchange = ExchangeNCL(
    tbmodels=tbmodel,
    atoms=atoms,
    efermi=0.0,
    magnetic_elements=magnetic_elements,
    kmesh=kmesh,
    emin=emin,
    emax=emax,
    nz=nz,
    exclude_orbs=exclude_orbs,
    Rcut=Rcut,
    ne=ne,
    description=description)
exchange.run()
```

In which the SislWrapper is a AbstractTB-like class which use sisl to read the Hamiltonian and overlap matrix from Siesta output.

## 5.2 Extend the output to other formats

The calculated magnetic interaction parameters, together with other informations, such as the atomic structure and some metadata, are saved in "SpinIO" object. By making use of it, it is easy to output the parameters to the file format needed. Some parameters, which cannot be calculated in TB2J can also be inputted so that they can be written to the files. The list of stored data is listed below, by using which it should be easy to write the output function as a member of the SpinIO class. A method write_some_format(path) can be implemented and called in the write_all method. Then the format is automatically written after the TB2J calculation.

**class** TB2J.io_exchange.**SpinIO**(*atoms*, *spinat*, *charges*, *index_spin*, *orbital_names={}*, *colinear=True*, *distance_dict=None*, *exchange_Jdict=None*, *exchange_Jdict_orb=None*, *dJdx=None*, *dmi_ddict=None*, *Jani_dict=None*, *biquadratic_Jdict=None*, *debug_dict=None*, *k1=None*, *k1dir=None*, *NJT_Jdict=None*, *NJT_ddict=None*, *damping=None*, *gyro_ratio=None*, *write_experimental=True*, *description=None*)

**Jani_dict = None**
The dictionary of anisotropic exchange. The vlaues are matrices of shape (3,3).

**atoms = None**
atomic structures, ase.Atoms object

**colinear = None**
If the calculation is collinear or not

**distance_dict = None**
A dictionary of distances, the keys are (i,j, R), where i and j are spin index and R is the cell index, a tuple of three integers.

**dmi_ddict = None**
The dictionary of DMI. the key is the same as exchange_Jdict, the values are 3-d vectors (Dx, Dy, Dz).

**exchange_Jdict = None**
The dictionary of $J_{ij}(R)$, the keys are (i,j, R), where R is a tuple, and the value is the isotropic exchange

**gyro_ratio = None**
Gyromagnetic ratio for each atom

**has_bilinear = None**
Whether there is anisotropic exchange term

**has_dmi = None**
Whether there is DMI.

**has_exchange = None**
whether there is isotropic exchange

**ind_atoms = None**
The index of atom for each spin.

**index_spin = None**
index of spin linked to atoms. -1 if non-magnetic

**classmethod load_pickle**(*path='TB2J_resutls'*, *fname='TB2J.pickle'*)

**plot_DvsR**(*ax=None*, *fname=None*, *show=False*)

**plot_JanivsR**(*ax=None*, *fname=None*, *show=False*)

**plot_JvsR**(*ax=None*, *color='blue'*, *marker='o'*, *fname=None*, *show=False*)

**plot_all**(*title=None*, *savefile=None*, *show=False*)

**spinat = None**
spin for each atom. shape of (natom, 3)

**write_Jq**(*kmesh*, *path*, ***kwargs*)

**write_all**(*path='TB2J_results'*)

**write_multibinit**(*path*)

**write_pickle**(*path='TB2J_results'*, *fname='TB2J.pickle'*)

**write_tom_format**(*path*)

**write_txt**(*path*)

**write_uppasd**(*path*)

**write_vampire**(*path*)

# CHAPTER 6

## Frequently asked questions.

- How to cite TB2J?

  See the reference section

CHAPTER 7

# Contributors

The main contributors to the TB2J package are:

- Xu He (mailhexu@gmail.com)
- Nicole Helbig
- Matthieu Verstraete
- Eric Bousquet

We welcome contributions. You can help us with - extend the input format to other codes, e.g. first principles or tight binding code. - extend the output to other spin dynamics or - extend the algorithm so it can be used to calculate other parameters. - test the validility of the methods implemented. - improve the documentation

Please also feel free to contact us if you think there are other things TB2J can help.

CHAPTER 8

# References

The implementation details of TB2J:

> Xu He, Nicole Helbig, Matthieu J. Verstraete, Eric Bousquet, TB2J: a python package for computing magnetic interaction parameters Computer Physics Communications, 107938 (2021).

The theoretical background for exchanges calculation is described in

- Initial idea of Green's function method of calculating J.

  Liechtenstein et al. J.M.M.M. 67,65-74 (1987), (aka LKAG)

- Isotropic exchange using Maximally localized Wannier function.

  Korotin et al. Phys. Rev. B 91, 224405 (2015)

- Full exchange tensor.

  Antropov et al. Physica B 237-238 (1997) 336-340

- Biqudratic term.

  S. Lounis, P. H. Dederichs, Phys. Rev. B 82 180404(R) (2010)

  Szilva et al, Phys. Rev. Lett. 111, 127204

- **Downfold method** I. V. Solovyev, Phys. Rev. B 103, 104428

# CHAPTER 9

# Release Notes

## 9.1 v0.6.6 September 1, 2021

Output the figure of J vs distance. Fix a bug when the orbitals are not grouped by atoms.

## 9.2 v0.6.4 August 9, 2021

Documentation of orbital decomposition. More concise orbital decompostion to the exchange with siesta2J.py. Allow to specify the orbitals used in the decomposition in siesta2J.py –element option.

## 9.3 v0.6.3 July 3, 2021

Revert qsolver to old version.

## 9.4 v0.6.2 May 27, 2021

**Enable reading structures from wannier .win file so –posfile is no more necessray.** –posfile option can still be used.

## 9.5 v0.6.1

Change the internal order of orbitals in noncollinear Tight-binding.

A script for building docker image has been added (Nikolas Garofil).

## 9.6 v0.6.0

Add TB2J_downfold.py script to deal with ligand spin contribution.

## 9.7 v0.5.0

Add Wannier input from banddownfolder package using –wannier_type=banddownfolder. Currently only collinear calculation supported.

## 9.8 v0.4.4 March 16, 2021

Allow parallel over k in tight binding eigen solver.

## 9.9 v0.4.3 March 11, 2021

Add Reference to TB2J paper.

## 9.10 v0.4.2 March 11, 2021

Fix a bug that the atoms scaled positions get wrapped. Fix a bug with consecutive parallel run in python mode.

## 9.11 v0.4.1 February 2, 2021

Use a Legendre path for the integration which is more stable and requires less poles(–nz). Memory optimization.

## 9.12 v0.4.0 February 1, 2021

Add –np option to specify number of cpu cores in parallel. Dependency on pathos is added.

## 9.13 v0.3.8 December 29, 2020

Add –output_path option to specify the output path.

## 9.14 v0.3.6 December 7, 2020

Use Simpson's rule instead of Euler for integration.

## 9.15 v0.3.5 November 3, 2020

Add –groupby option in wann2J.py to specify the order of the basis set in the hamiltonian.

## 9.16 v0.3.3 September 12, 2020

- Use collinear exchange calculator for siesta-collinear calculation, which is faster.

## 9.17 v0.3.2 September 12, 2020

- **add –use_cache option to reduce the memory usage by storing the Hamiltonian** and eigenvectors on disk using memory map.

## 9.18 v0.3.1 September 3, 2020

- A bug in the sign of the magnetization along y in Wannier and OpenMX mode is fixed.

## 9.19 v0.3 August 31, 2020

- A bug in calculation of anisotropic exchange is fixed.
- add TB2J_merge.py for merging DMI and anisotropic exchange from calculations with different spin orientation or structure rotation.
- Improvement on output txt file.
- An interface to OpenMX (TB2J_OpenMX) is added in a separate github under GPLv3. at https://github.com/mailhexu/TB2J-OpenMX
- Many improvement and bugfixes

## 9.20 v0.2 2020

- Moved to github
- DMI and anisotropic exchange
- Magnon band structure (For FM and single magnetic specie)
- Siesta Input
- Documentation on readthedocs

## 9.21 v0.1 2018

- Initial version on gitlab.abinit.org
- Isotropic exchange

- Wannier function as input
- Interface with Multibinit, Tom's ASD, and Vampire

CHAPTER 10

# Indices and tables

- genindex

# Index